To read the article online, visit http://www.4GuysFromRolla.com/articles/102208-1.aspx

Examining ASP.NET's Membership, Roles, and Profile - Part 13

By Scott Mitchell

Introduction

ASP.NET's forms-based authentication system in tandem with the Membership API and Login Web controls make it a cinch to create a user store, create user accounts, and allow visitors to log into the site. What's more, with little effort it's possible to define roles, associate user accounts with roles, and determine what functionality is available based on the currently logged in user's role (see Part 2). Many ASP.NET sites that use Membership have an Admin role, and users in that role are granted certain functionality not available to non-Admin users. Consider an online store - Admin users might be able to manage inventory, whereas the only way normal members could interact with the inventory was by adding items to their shopping cart.

I was recently working with a client who had an interesting request: he needed the ability for Admin users to be able to log into the site as another user, and perform actions as if that other person had logged in herself. Returning to the online store example, imagine that some customers periodically phone in their order, or mail or fax in an order form. An Admin, receiving this order, could then log into the site as that customer and place the order on the customer's behalf.

This article shows how to allow an Admin user to log into a Membership-based website as another user, and includes a complete working demo available for download at the end of the article. Read on to learn more!

Logging In As Another User

The ASP.NET Login control provides the user interface and logic for logging into a Membership-enabled website. The Login control presents two textboxes, one for the Username and one for the Password, along with a Login button. Clicking the Login button causes a postback during which the Login control attempts to validate the supplied credentials (the username and password) against the Membership system. If the credentials are valid, the user is logged into the system by means of a forms authentication ticket, which is a cookie that is saved to the user's browser that serves as the identity for the request. This cookie is sent from the browser to the website on subsequent requests and is the means by which a user remains "logged on" as they visit different pages on the site.

For certain websites it may make sense to allow Admin users to log into the site as another user in the system. (See "The Pros and Cons of Logging In As Another User" sidebar for a discussion that weighs the benefits of such a service against the potential harms.) If the Admin user knew the password of the user he wanted to log in as, then logging in as another user would be straightforward - the Admin user would simply enter the name and password of the user to log in as in the appropriate textboxes in the login page. However, it is usually the case that the Admin user does not know, and cannot find out, another user's password. By default, passwords in the SQL Server-based Membership system store the passwords in a hashed format. Because it is impossible to translate from the hashed format back to the original password, the Admin user cannot determine a user's password (unless the user divulges that information).

What's important to keep in mind is that the Login Web control is used simply to validate the supplied credentials and then create a corresponding forms authentication ticket, and that the forms authentication ticket is what identifies the visitor. Therefore, it is possible to create an ASP.NET page that "logs in" a user without knowing their password - all you have to do is create a valid forms authentication ticket with the username of the person you want to log in as. (After all, that's all the Login control does after it validates the supplied credentials.)

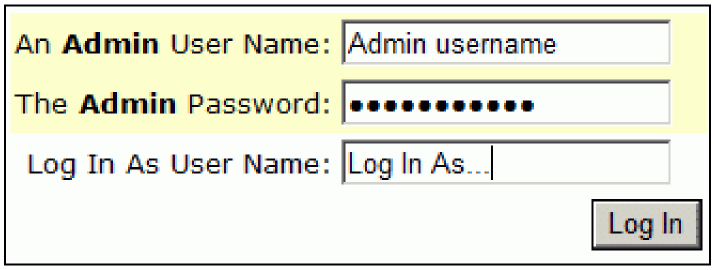
This article shows how to create an alternative login page that consists of three textboxes:

- One for the Admin user's username,
- One for his password, and
- One for the name of the user to log in as

In addition to these textboxes the page includes a Button Web control that, when clicked:

- Validates the Admin username and password,
- Ensures that the valid credentials are for an Admin user, and
- Ensures that the username of the user to log in as exists in the Membership system

If all of these checks pass then the page creates a forms authentication ticket for the username of the user to log in as. The net effect is that an Admin user can visit the page, enter his credentials, type in another user's username, and then log in as that user.



An **Admin** User Name: Admin username

The **Admin** Password: ●●●●●●●●●●●●

Log In As User Name: Log In As...

Creating the User Interface

The user interface shown in the screen shot above is implemented as an HTML `<table>` with five rows in total - one row for each of the three TextBox controls, one for the Login Button control, and one that contains displays a message upon log in failure. In addition to the three TextBox controls the page contains three RequiredFieldValidator controls to ensure that the user supplies values for each of the inputs. (For more information on ASP.NET's validation controls, see Form Validation with ASP.NET - It Doesn't Get Any Easier!.)

You can build the above user interface by hand. A quicker way, though, is to add a Login control to the page and then turn it into a template (one of the options from the control's smart tag when viewed in the Designer). Doing so generates an HTML `<table>` within the Login control's declarative syntax. I then deleted the Login control tags (`<asp:Login>` and `</asp:Login>`), leaving the markup. Finally, I added a new table row for the "Log In As User Name" UI and removed the "Remember Me" CheckBox row (because I do not want to allow an Admin user to log on as another user and have that remembered across browser restarts).

Here's a snippet of the HTML and Web controls used for the user interface. Here you see the HTML `<table>` and the first row that contains the TextBox for the Admin user's username.

```
<table border="0" cellpadding="2" cellspacing="0">
   <tr class="AdminUserPrompt">
      <td align="right">
         <asp:Label ID="AdminUserNameLabel" runat="server"
AssociatedControlID="AdminUserName">An <b>Admin</b> User Name:</asp:Label>
      </td>
      <td>
```

```
         <asp:TextBox ID="AdminUserName" runat="server"></asp:TextBox>
         <asp:RequiredFieldValidator ID="AdminUserNameRequired" runat="server"
            ControlToValidate="AdminUserName" ErrorMessage="The Admin User Name is
required."
            ToolTip="The Admin User Name is required." ValidationGroup="LogInAs">*
</asp:RequiredFieldValidator>
      </td>
   </tr>
   ...
</table>
```

Logging in the User

After entering the Admin user's credentials, the name of the user to login as, and clicking the Login button, a postback occurs. In the Login Button's `Click` event handler we need to: validate the Admin user's credentials; assure that the Admin user is actually in the Admin role; and verify that the username to log in as exists in the system. If all that checks out, we need to create a forms authentication ticket for the user. The following code shows the `Click` event handler that handles this logic:

```
Protected Sub LoginButton_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles LoginButton.Click
   'Make sure that the Admin username & password are valid
   If Membership.ValidateUser(AdminUserName.Text, AdminPassword.Text) Then
      'Yes, this username/password is good, but is this user in the Admin role?
      If Roles.IsUserInRole(AdminUserName.Text, "Admin") Then
         'Great, this user is in the Admin role! Now, is the username to login as a valid
user?
         Dim LogInAsUser As MembershipUser = Membership.GetUser(LogInAsUserName.Text)

         If LogInAsUser IsNot Nothing Then
            'Yes, this user is valid! Great, let's log in as that user!
            FormsAuthentication.RedirectFromLoginPage(LogInAsUserName.Text, False)
         Else
            'The Admin username/password is kosher, but the user to log in as was not
found
            FailureText.Text = String.Format("The user {0} does not exist in the
Membership database.", AdminUserName.Text)
         End If
      Else
         'The user credentials for Admin user are valid, but the user is not an Admin
         FailureText.Text = "Only Admins can log into the site as another user. To login
as yourself, please visit the standard Login page."
      End If
   Else
      'The Admin username/password are invalid
      FailureText.Text = "Your login attempt was not successful. Please try again."
   End If
End Sub
```

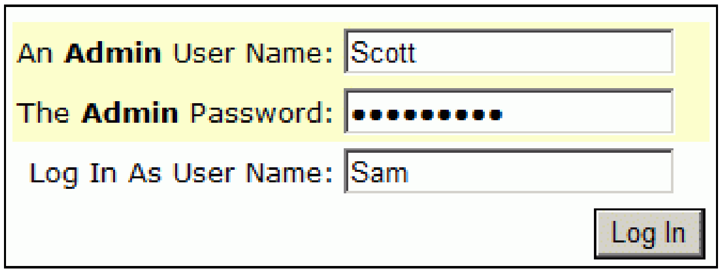
The code starts by calling the `Membership.Validate(`*`username`*`, `*`password`*`)` method, which returns True if the supplied credentials are valid, False otherwise. The username and password entered into the Admin username and password TextBox controls are passed in as the *username* and *password* parameters to this method. If this returns True then we next use the Roles API to ensure that the user is a member of the Admin role. If it returns False, the message "Your login attempt was not successful. Please try again" is displayed.

To determine if the Admin user credentials supplied belong to a user in the Admin role, the Roles API is used. A call to `Roles.IsUserInRole(`*`username`*`, `*`role`*`)` returns a Boolean value indiciating whether *username* exists in the role *role*. If the user is indeed an Admin then the last step is to check whether the username of the user to log in as is indeed valid. If the user is *not* in the Admin role then the message "Only Admins can log into the site as another user. To login as yourself, please visit the standard Login page" is displayed.

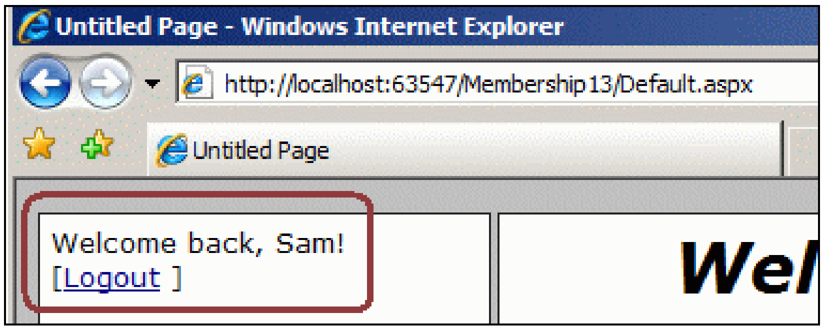
To determine if the username of the user to log in as is valid we call the `Membership.GetUser(`*`username`*`)`, which returns a `MembershipUser` object if the username is valid. If the user is not found then it returns `Nothing` (`null` in C#). If the user account exist we log in as that user by creating a forms authentication ticket with that user's identity. The `FormsAuthentication` class contains methods for creating and managing forms authentication tickets. The `RedirectFromLoginPage(`*`username, persistentCookie`*`)` method creates a forms authentication ticket for user *username*, adds it to the visitor's `Cookies` collection, and then redirects the user to the appropriate page (the URL specified in the `ReturnUrl` querystring value, if present; `Default.aspx` otherwise).

If the user to log in as is not found in the system the message "The user *username* does not exist in the Membership database" is displayed.

That's all there is to it! With this code in place an Admin user can visit the site, enter his credentials and the name of another user on the site, and then log in as that user. The following screen shots show this interaction. In the first screen shot, Admin user Scott enters his credentials and the name of another user, Sam. Clicking the Login button causes a postback and Scott is signed in as Sam and redirected to the homepage.


An **Admin** User Name: Scott
The **Admin** Password: ••••••••••
Log In As User Name: Sam

As you can see, Scott has logged in as Sam. For all intents and purposes, Scott *is* Sam. Scott will have the same user experience as Sam would have, if she was logged on. Also, any actions performed by Scott will be as if they were performed by Sam.


Untitled Page - Windows Internet Explorer
http://localhost:63547/Membership13/Default.aspx
Untitled Page
Welcome back, Sam!
[Logout]

The Pros and Cons of Logging In As Another User

I've helped several small businesses move their paper-based business processes to a more modern electronic-based system, many of which have resulted in websites that have used the ASP.NET Membership system. Many of these businesses have requested the ability for Admin users to be able to log on as other users, and each have their own reasons. As noted in the Introduction, one business still had customers that faxed in orders, so they wanted to be able to place orders through the online system as that user who faxed in the order. Another client of mine had users who spent most of their day on a jobsite and recorded hours and other information into the system at the end of the day. If there was a data entry error or some missing piece of information, it was convenient for the secretary to be able to go in and fix the problem or add the data immediately than to wait for the user to return to the home office.

Allowing Admin users to log in as any other user has the advantage that the Admin users can very easily enter the system and make changes to another user's data or information. There's no need to coorindate with the user (who may be on a jobsite, for instance). Of course, this approach is a bit hamfisted. In a perfect world, there would be a set of administrative pages that would enable Admin users to make these changes as needed. But when you're working with a client on a tight budget and their choice is, "Allow users to log in as any other user to make changes," or, "Spend several days creating administration pages," the first choice is more attractive. It also has the benefit that it makes it easier for an Admin to diagnose a bug that is happening for a specific user because of user-specific data. That is, the Admin can log in as that user who is getting the error and is able to replicate it and test later to ensure that it's been fixed properly.

The downside to letting Admin users log in as other users is that while the Admin is logged in as another user, they can accidentally (or purposefully) wreck havoc. The Admin might forget that she's logged in as another user, and go and enter data specific to her, not realizing that she's entering that data for the user she is logged on as. Another problem is that there is no auditing capabilities, at least not how the feature has been implemented thus far. Imagine that a user asks, "Why does my order contain XYZ? I didn't order XYZ." You look in the database and see that the customer did indeed order XYZ and refuse a refund. But wait, did the customer really order that or was an Admin user logged on as that customer and placed that order? There's no way to tell!

If you are going to allow Admin users to log on as any other user I would, at minimum, encourage you to place very strict limits as to what users are Admins (or what users can use this "log on as" feature). I would also encourage you to extend the functionality we discussed above to remember the Admin user when he logs in as another user. The final section of this article discusses how to do this!

Storing the Username of the Admin User Who Logged On As Another User

The `FormsAuthentication.RedirectFromLoginPage` method automatically creates a forms authentication ticket and adds it to the visitor's `Cookies` collection. The `FormsAuthentication` class also contains a `GetAuthCookie(`*`username, persistCookie`*`)` method that creates the forms authentication ticket and returns it, but does not add it to the `Cookies` collection, nor does it redirect the user to the appropriate page.

Along with indicating the name of the authenticated user, the forms authentication ticket can also hold additional developer-defined data. Rather than creating the ticket via `FormsAuthentication.RedirectFromLoginPage` we can instead use `FormsAuthentication.GetAuthCookie` to create the authentication ticket and then add the name of the Admin user to the ticket. This way the Admin is still able to log on as another user, but we can determine the name of the Admin user. This information can be used to display a message to alert the Admin user that they are logged on as another user. It can also be used for auditing purposes to record if an Admin logged on as the user made a change. (To see what I mean, consider an online store. There is an `Orders` table that links an order to a customer. This table could be expanded to include a column named `AminLoggedOnUserId`. If a user logged on to the site and placed an order, the `Orders` record's `AminLoggedOnUserId` column would have a value of `NULL`, but if Admin user Scott logged on as Sam to place the order, the `Orders` record's `AminLoggedOnUserId` column would store Scott's `UserId` value, making it clear that Scott placed the order on Sam's behalf.)

To use this functionality, replace the call to `FormsAuthentication.RedirectFromLoginPage` with the following code:

```
' Create the cookie that contains the forms authentication ticket
Dim authCookie As HttpCookie = FormsAuthentication.GetAuthCookie(LogInAsUserName.Text,
False)
```

```
' Get the FormsAuthenticationTicket out of the encrypted cookie
Dim ticket As FormsAuthenticationTicket = FormsAuthentication.Decrypt(authCookie.Value)

' Create a new FormsAuthenticationTicket that includes our custom User Data
Dim newTicket As FormsAuthenticationTicket = New
FormsAuthenticationTicket(ticket.Version, ticket.Name, ticket.IssueDate, _
                                             ticket.Expiration, ticket.IsPersistent,
AdminUserName.Text)

' Update the authCookie's Value to use the encrypted version of newTicket
authCookie.Value = FormsAuthentication.Encrypt(newTicket)

' Manually add the authCookie to the Cookies collection
Response.Cookies.Add(authCookie)

'Return the user to the homepage
Response.Redirect("~/Default.aspx")
```

The above code starts by calling `FormsAuthentication.GetAuthCookie` to create the forms authentication cookie for the user who the Admin user wants to log in as. This cookie is actually encrypted so as to protect its contents as the cookie travels over the wire from the visitor's browser to the web server. Therefore, we need to descrypt it before we can work with it; this is accomplished by using the `FormsAuthentication.Decrypt` method. Next, we create a *new* forms authentication ticket, this time indicating that the Admin user's username should be included in the ticket's contents. This new authentication ticket is encrypted and added to the `Cookies` collection. Finally, the user is redirected to the homepage (`Default.aspx`).

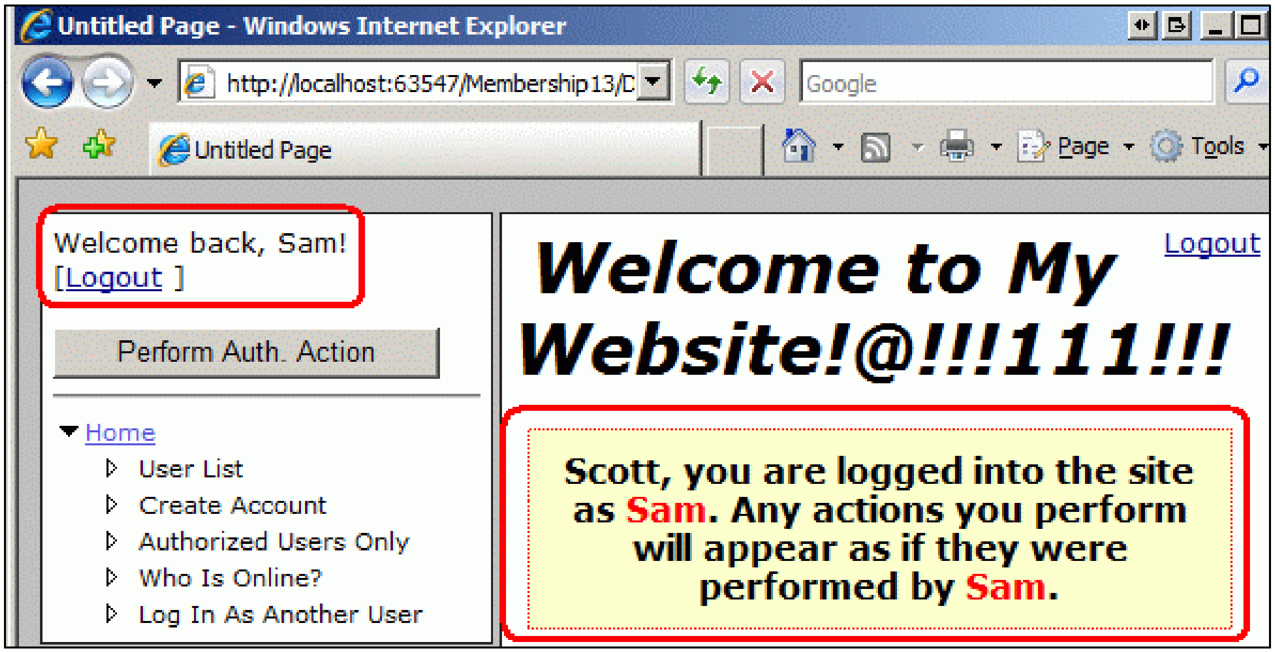
At this point whenever an Admin user logs on as another user the resulting forms authentication ticket identifies the user to log on as, but it also includes the Admin user's name (in the user data portion of the ticket). We can programmatically retrieve the Admin user's name using the following code:

```
If Page.User.Identity IsNot Nothing AndAlso TypeOf Page.User.Identity Is FormsIdentity
Then
   Dim ident As FormsIdentity = CType(Page.User.Identity, FormsIdentity)
   Dim ticket As FormsAuthenticationTicket = ident.Ticket
   Dim AdminUserName As String = ticket.UserData

   If Not String.IsNullOrEmpty(AdminUserName) Then
      'An Admin user is logged on as another user...
      'The variable AdminUserName returns the Admin user's name
      'To get the currently logged on user's name, use Page.User.Identity.Name
   Else
      'The user logged on directly (the typical scenario)
   End If
End If
```

For more information on how to programmatically add and retrieve user data to a forms authentication ticket, see Forms Authentication Configuration and Advanced Topics.

I've updated the master page (`~/MasterPage.master`) to include a Panel that displays a prominent message if an Admin user is currently logged in as another user. This message is shown only if an Admin user is logged on as another user; if a user logs on as himself, this message is not displayed.



Scott, you are logged into the site as Sam. Any actions you perform will appear as if they were performed by Sam.

Conclusion
Forms-based authentication, the Membership system, and ASP.NET's Login controls make it easy to build a web application that supports user accounts. Unfortunately, there are no built-in tools to allow administrative users to log into the site as another user in the system. The good news is that such functionality can be implemented with a couple dozen lines of code, as evidenced in this article. Be sure to download the demo application to see this functionality in action.

Happy Programming!

- By Scott Mitchell

Further Reading

- Forms Authentication, Authorization, Membership, and Roles Tutorials (includes VB & C# versions!)
- Forms Authentication Configuration and Advanced Topics (VB) (C# Version)
- Form Validation with ASP.NET - It Doesn't Get Any Easier!
- Dissecting Forms Authentication

Attachments

- Download the code used in this article

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 13
Article Author:	Scott Mitchell
Published Date:	October 22, 2008
Article URL:	http://www.4GuysFromRolla.com/articles/102208-1.aspx